Network Working Group   P. Rawat
Internet-Draft   J-M. Bonnin
Expires: September 27, 2009   TELECOM Bretagne
                                                        A. Minaburo
                                                        JCP-Consult
                                                            E. Paik
                                                                 KT
                                                     March 26, 2009

Tunneling Header Compression (TuCP) for Tunneling over IP
draft-rawat-mext-hc-tunneling-00

Status of this Memo

   This Internet-Draft is submitted to IETF in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF), its areas, and its working groups. Note that
   other groups may also distribute working documents as Internet-
   Drafts.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   The list of current Internet-Drafts can be accessed at
   http://www.ietf.org/ietf/1id-abstracts.txt.

   The list of Internet-Draft Shadow Directories can be accessed at
   http://www.ietf.org/shadow.html.

   This Internet-Draft will expire on September 27, 2009.

Copyright Notice

Abstract

   The IP tunneling mechanisms have important applications in network
   solutions and are widely used in numerous contexts such as security
   (VPN), IPv4 to IPv6 transition, and mobility support (MobileIP and
   NEMO). However, these tunneling mechanisms induce a large overhead
   resulting from adding several protocol headers in each packet. This
   overhead deteriorates performance on wireless links which are scarce
   in resources.

   Header compression methods are often used on connection oriented
   communication (e.g., UMTS networks) to reduce the overhead on the
   wireless part. These header compression methods can be used on
   tunnel headers to reduce the protocol header overheads, independent
   of the payload type. Although, several header compression methods
   exist, the header compression profiles defined by them are not
   adapted to the characteristics of IP tunneling. This document
   specifies a tunneling header compression protocol for IP tunneling
   mechanisms.

Requirements Language

   The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
   "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
   document are to be interpreted as described in RFC 2119 [BCP].

Table of Contents

1. Introduction

   Today's Internet uses wide variety of IP tunnels over wired links,
   which are rich in bandwidth as well as over wireless links which are
   low in bandwidth. Moreover, resources vary in mobile networks due to
   radio conditions. IP tunneling is widely used in several contexts.
   IP tunneling has been used for many years by ISPs to offer VPNs with
   private addresses. IP-in-IP tunneling, the simplest IP tunneling
   method, is used in IP mobility protocol to provide IP mobile node
   with mobility support and security in conjunction with IPSec. Some
   tunneling methods are used in order to build an overlay network for
   transition purposes, i.e., to pass through an IPv4 cloud to reach the
   IPv6 Internet. One such example is using [L2TP] for providing IPv6
   over IPv4 only access [I-D.softwire-hs]. As we connect to an ISP and
   since we often have to traverse a NAT, these methods tend to use a
   transport protocol such as [TCP] or [UDP], for example with L2TP.
   The latter allows to extend a [PPP] connection through the Internet
   to the Network Access Server of the ISP.

   These tunneling mechanisms induce a large overhead resulting from
   adding several protocol headers in each packet, for instance at least
   IP/UDP/L2TP/PPP headers in the above example. Moreover, this header
   overhead can be high on wireless links which have scarce resources.
   IP tunneling involves encapsulating a packet within another packet,
   both of which support the same or different protocols. This requires
   adding a new stack of headers to the tunneled packet, hence
   increasing the size of the headers. Since tunnels may be set up to
   pass through links with low bandwidth and scarce resources such as
   wireless links. In that case the increase in header size will
   consume more bandwidth and waste the resources especially when
   headers may contain some redundancy in their fields.

   In order to reduce the tunneling header overhead and save the link
   resources, header compression mechanisms may be used independently of
   the payload type. Header compression mechanisms can reduce the size
   of a header by removing redundancies from the header fields.
   However, much of the existing work in header compression [IPHC],
   [ROHC], [ECRTP], [CTCP] centers around the compression of inner
   headers (for example, IP, UDP and [RTP]) of the IP packet passing
   through the tunnel and does not deal with the compression of the
   outer headers used by tunneling mechanisms (for example, L2TP and
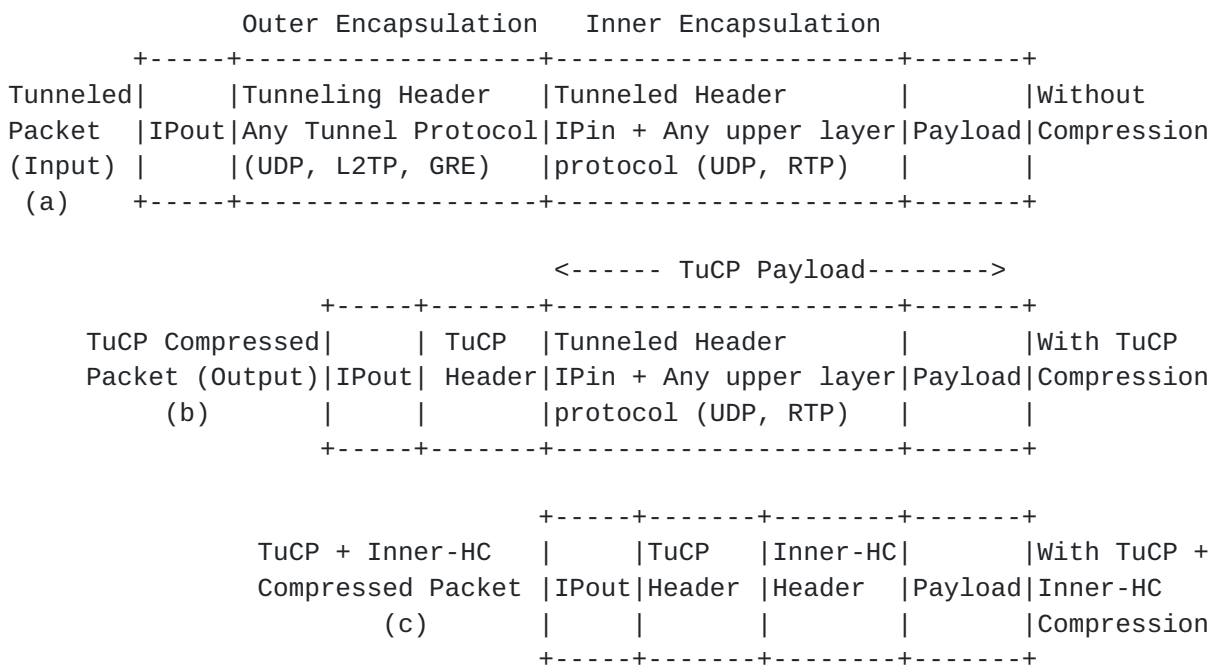   [GRE]).

   This document presents a tunneling header compression protocol, TuCP
   (Tunneling Compression Protocol), that can be used over IP tunneling
   mechanisms. It can compress headers of various tunneling protocols
   such as UDP, L2TP, GRE etc. In addition, TuCP provides a solution
   for the packet reordering or out-of-sequence problem (in tunneling)

and thus it extends the usage of existing header compression
mechanisms such as ROHC from point to point links to the IP tunnels
passing over the Internet.

2. Motivations

IP tunneling consists of inner and outer encapsulations as shown in
Figure 1. The tunneled protocol gives the inner encapsulation and
the tunneling protocol is the outer encapsulation. In mobile
networks ([MobileIP] and [NEMO]), the overhead due to inner IP
encapsulation can be reduced using an existing header compression
mechanism. However, the tunnel itself has overhead due to its IP
header (IPout) and the tunneling header. The legacy header
compression mechanisms do not define a profile to compress tunneling
headers. Furthermore, in order to use them for outer encapsulation,
it will be required to modify them to take into account tunneling.


```
              Outer Encapsulation   Inner Encapsulation
        +-----+------------------+--------------------+-------+
Tunneled|     |Tunneling Header  |Tunneled Header     |       |Without
Packet  |IPout|Any Tunnel Protocol|IPin + Any upper layer|Payload|Compression
(Input) |     |(UDP, L2TP, GRE)  |protocol (UDP, RTP) |       |
 (a)    +-----+------------------+--------------------+-------+

                      <------ TuCP Payload-------->
              +-----+-------+--------------------+-------+
   TuCP Compressed|     | TuCP |Tunneled Header     |       |With TuCP
   Packet (Output)|IPout| Header|IPin + Any upper layer|Payload|Compression
        (b)       |     |      |protocol (UDP, RTP) |       |
              +-----+-------+--------------------+-------+

              +-----+-------+--------+-------+
      TuCP + Inner-HC  |     |TuCP  |Inner-HC|       |With TuCP +
      Compressed Packet |IPout|Header |Header  |Payload|Inner-HC
            (c)         |     |      |        |       |Compression
              +-----+-------+--------+-------+
```

        Inner-HC = ROHC, CTCP, ECRTP, IPHC, VJCOMP (see section 7)

        Figure 1: Inner and Outer Encapsulations in IP Tunneling

Often, the protocol stack is much complex, for example, in the case
of IP tunneling it can use L2TP protocol and thus it will include
UDP/L2TP/PPP headers (stack). Hence, the global stack will be IPout/

UDP/L2TP/PPP/IPin/UDP/RTP, where IPout is the outer IP header and
IPin implies the inner IP header. The only header which is necessary
to reach the tunnel endpoint is the IPout header, therefore we can
compress all other headers present in the packet.

The existing header compression mechanisms only compress the inner IP
encapsulation such as IPin/UDP/RTP. Therefore, there is a lack of a
method to compress the outer encapsulation, which is UDP/L2TP/PPP
encapsulation in the above example. It should be noted that the
outermost IP header of the tunnel SHOULD NOT be compressed as it is
required by the intermediate routers to route the packet to the
tunnel endpoint.

Moreover, present header compression mechanisms do not deal with the
case of nested tunnels even if supplementary headers used for inner
tunnels are useless for the outermost tunnel packet routing purpose.
Furthermore, most of these compression algorithms have been defined
to work in a point to point link where reordering of packets does not
take place. Tunneling over IP does not guarantee the ordered arrival
of packets to the tunnel endpoint; hence these mechanisms are not
very effective in the case of tunneling.

To address these issues, this document introduces a header
compression mechanism for IP tunneling; TuCP (Tunneling Compression
Protocol). TuCP is defined to compress the outer encapsulation when
tunneling is used (see Figure 1 (b)). It compresses the tunneling
header overhead into 3-5 bytes. TuCP is extensible to general
tunneling headers compression. In addition, TuCP provides a solution
for the packet reordering problem so that legacy compression
mechanisms, such as ROHC can also be used to compress the inner
encapsulation as shown in Figure 1 (c). TuCP is much simpler than
ROHC since tunneling headers are mostly static and do not change from
one packet to another.

[3](#). **Overview of Header Compression with TuCP**

Header compression can be applied on tunneling headers because there
is significant redundancy between header fields between consecutive
packets belonging to the same packet flow. TuCP removes the
redundant header information by classifying the tunneling header
fields into static and dynamic fields depending on their changing
characteristics. TuCP installs a compressor and decompressor entity
at each tunnel endpoint. The TuCP compressor first sends both the
static and dynamic fields to establish the complete tunneling header
information (static and dynamic information) at the TuCP
decompressor. After that, the compressor sends only the dynamic
fields to the decompressor and the static fields are not sent.

This reduces the overhead due to redundant header information. For
example, an IP/UDP/L2TP/PPP packet consists of a 20 bytes IPv4
header, an 8 bytes UDP header, a 6 bytes L2TP header (maximum L2TP
header is 16 bytes), and a 4 bytes PPP header. Thus, the total
header transmitted over wireless link has a minimum length of 38
bytes, which will further increase to 58 bytes in presence of IPv6
(40 bytes). But most of the fields are static and do not change
between two successive packets belonging to the same tunnel flow (see
section 4). Thus, it is not REQUIRED to send static information to
avoid needless burden, especially on wireless links.

4. Classification of Tunneling Header Fields

This section gives a general classification of UDP, L2TP, PPP and GRE
tunneling header fields as shown in Figures 2, 3, 4 and 5,
respectively.

TuCP classifies the header fields into the following three classes:

INFERRED (NOT SENT): These fields contain values that can be inferred
from other values, and thus they are easily compressed by the
compression scheme. The values in these fields are not sent as they
can be inferred.

STATIC: These fields contain values that remain constant throughout
the lifetime of the flow. Static information is communicated only
once. In this document, the terminology "flow" refers to a set of
packets having the same values in their STATIC fields.

DYNAMIC: These fields vary randomly or in a predictable pattern
within a limited range.

```
    +-----------------+-------------+-----------+
    | Header Field    |Classification| Size (bits)|
    +-----------------+-------------+-----------+
    | Source Port     |   STATIC    |    16     |
    | Destination Port|   STATIC    |    16     |
    | Datagram Length |   INFERRED  |    16     |
    | Checksum        |   DYNAMIC   |    16     |
    +-----------------+-------------+-----------+
```

Figure 2: UDP Header Fields

The header size of UDP is 8 bytes when UDP checksum is enabled. In
IPv6, UDP checksum must be enabled.

```
+-----------------+--------------+-----------+
| Header Field    |Classification|Size (bits)|
+-----------------+--------------+-----------+
| T flag          |  STATIC      |     1     |
| L flag          |  STATIC      |     1     |
| S flag          |  STATIC      |     1     |
| O flag          |  STATIC      |     1     |
| P flag          |  STATIC      |     1     |
| Reserved        |  STATIC      |     7     |
| Version         |  STATIC      |     4     |
| Length (opt)    |  INFERRED    |    16     |
|Tunnel ID (opt)  |  STATIC      |    16     |
|Session ID (opt) |  STATIC      |    16     |
| Ns (opt)        |  DYNAMIC     |    16     |
| Nr  (opt)       |  STATIC      |    16     |
|Offset Size (opt)|  DYNAMIC     |    16     |
|Offset Pad (opt) |  INFERRED    |    16     |
+-----------------+--------------+-----------+
```

                   Figure 3: L2TP Header Fields

   The L2TP header is of 6-16 bytes. Minimum header size of L2TP is 6
   bytes.

```
+------------+--------------+-----------+
|Header Field|Classification|Size (bits)|
+------------+--------------+-----------+
|   Address  |   STATIC     |    8      |
|   Control  |   STATIC     |    8      |
|   Protocol |   STATIC     |    16     |
+------------+--------------+-----------+
```

                   Figure 4: PPP Header Fields

   NOTE: There are additional headers in PPP like Flag, Information,
   Padding, FCS which are not considered here. This draft considers the
   minimum PPP header (4 bytes) used in the IP/UDP/L2TP/PPP
   encapsulation.

```
+----------------+-------------+----------+
| Header Field   |Classification|Size (bits)|
+----------------+-------------+----------+
| C flag         |   STATIC    |    1     |
| Blank          |   STATIC    |    1     |
| K flag         |   STATIC    |    1     |
| S flag         |   STATIC    |    1     |
| Reserved0      |   STATIC    |    9     |
| Version        |   STATIC    |    3     |
| Protocol Type  |   STATIC    |   16     |
| Checksum (opt) |   DYNAMIC   |   16     |
| Reserved1 (opt)|   STATIC    |   16     |
| Key (opt)      |   STATIC    |   32     |
| Sequence Number|   DYNAMIC   |   32     |
|  (opt)         |             |          |
+----------------+-------------+----------+
```

Figure 5: GRE Header Fields

The GRE header is of 4-16 bytes. Minimum header size of GRE is 4
bytes.

The above figures show that most of the header fields in the
tunneling headers can be compressed as they do not vary. However a
small number of fields, (e.g., Checksum, Sequence Number) vary and
some of them, for example, Sequence Number vary in a predictable
manner. Hence, by sending only static fields' information initially
and utilizing dependencies and predictability for dynamic fields,
header size can be significantly reduced for most packets.

As the static fields are constant values, (for example, source and
destination addresses, ports), for a (tunnel) flow it would be enough
to send these fields initially to the destination. Once these static
fields are received at the destination, there would be no need to
send them again in every packet. As long as their values are stored
at the endpoints of the tunnel, they can be used again for each
packet belonging to the same tunnel.

On the contrary, dynamic fields (for example, sequence numbers,
checksum) of the same (tunnel) flow will show variations in their
values from one packet to another. These changes may follow a
pattern. Header fields whose values are always incrementing, such as
counters, can be predicted at the destination by keeping a reference
value. Whereas, dynamic fields with values that show random changes
and do not follow any set pattern, will have to be sent as they are
for each packet.

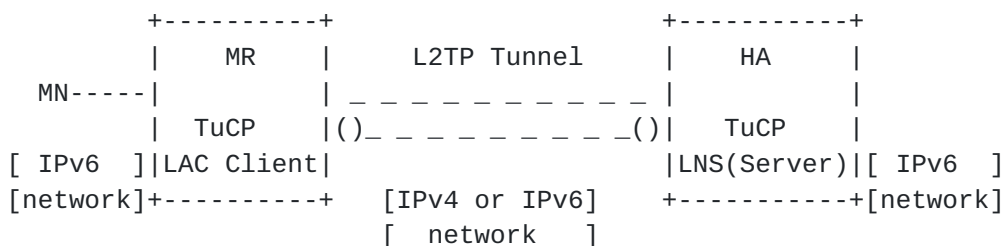There are many optional fields (opt) in some of the tunneling

headers, for example, L2TP and GRE (Figure 3 and 5). Whenever the
optional fields are present in the tunneling header, they SHOULD be
treated depending on the type of header classification. The Sequence
Number when present in L2TP and GRE headers may be encoded for
achieving further compression. It should be noted that in case of
L2TP, if the L2TP offset padding is non-zero, the compressor MUST
send it otherwise UDP checksum will fail.

In the GRE header, the bits left blank (see figure 5) and the
reserved bits are set to zero when sending and ignored when received.
Similarly, in the L2TP header, the reserved bits are set to zero when
sending and ignored when received.

5. TuCP protocol

In the protocol implementation, TuCP is implemented as a layer below
the IP layer. The TuCP protocol works in the following way. At the
sender side, when the TuCP compressor gets the encapsulated IP packet
(IPin), it compresses the headers added by the tunneling protocol
(e.g., L2TP) except IPout (see Figure 1) which is required for
routing purposes. At the receiver side, when the TuCP decompressor
gets the compressed (TuCP) packet, it decompresses the packet and
gives the decompressed packet to the IP layer. In the actual
protocol stack, TuCP operates over IP layer, as it does not compress
IP. It only changes the "Protocol" field in the IPv4 header ("Next
Header" field in the IPv6 header) to indicate the presence of TuCP as
the next header in the stack.

The following figure shows a NEMO network scenario, where TuCP
compression and decompression are applied at the tunnel endpoints, MR
(Mobile Router) and HA (Home Agent).

```
            +----------+                     +-----------+
            |    MR    |     L2TP Tunnel      |    HA     |
     MN-----|          | _ _ _ _ _ _ _ _ _ _  |           |
            |  TuCP    |()_ _ _ _ _ _ _ _ _()|   TuCP     |
   [ IPv6  ]|LAC Client|                     |LNS(Server)|[ IPv6  ]
   [network]+----------+   [IPv4 or IPv6]     +-----------+[network]
                          [  network   ]

                         NEMO network

              L2TP Encapsulation: PPP o L2TP o UDP o IP
```

Figure 6: NEMO Network Scenario with TuCP

At the compressor side, TuCP compression is applied to the packet

after the tunneling header has been added and before the packet is
sent into the tunnel, i.e., before the routing decision is taken. At
the decompressor side of the tunnel, TuCP decompression is applied
once the TuCP packet is received and before it is passed to the
decapsulation entity of the tunnel.

5.1. TuCP Profiles

A header compression profile specifies how to compress the headers of
a certain type of packet. TuCP defines various profiles called TuCP
profiles for compression of different tunneling headers.

TuCP defines five profiles; profile 0, 1, 2, 3 and 4 as shown below
in Figure 7. Further compression profiles MAY be defined in TuCP as
it is extensible to general tunneling headers compression. We can
use the TuCP profiles together with any header compression mechanism
to reduce the protocol header size.

```
         +-----------+--------------------+
         | Profiles  |   Tunnel Headers   |
         +-----------+--------------------+
         | Profile 0 | No tunneling header|
         | Profile 1 | UDP                |
         | Profile 2 | UDP/L2TP/PPP       |
         | Profile 3 | L2TP/PPP           |
         | Profile 4 | GRE                |
         +-----------+--------------------+
```

Figure 7: TuCP Profiles

Profile 0: This profile is defined for sending uncompressed mobile IP
(IP/IP) packets. This is the most basic profile which is used when
there are no tunneling headers, this profile adds a TuCP header to
the original (input) packet at the compressor side. This TuCP header
will be used for CRC and TSN fields (section 5.2) to be able to
detect packet damage, loss or reordering at the decompressor side.
This makes it possible to take appropriate action at the decompressor
if packets arrive out of order. This profile can treat any kind of
tunnel packets. A specific use of this profile will be in a scenario
when TuCP is used in conjunction with another header compression
scheme e.g., ROHC. In this scenario, the tunneling header (outer IP
header) is not compressed by TuCP as it is used for routing purposes,
but the tunneled header (inner IP header) is optionally compressed by
ROHC or any other header compression scheme.

Profile 1: For UDP compression when the tunnel is UDP based. This
profile can be used for basic UDP based tunnels, for example, to
compress UDP header when the protocol header stack is IP/UDP.

   Profile 2: For L2TP based tunnels. This profile can be used to
   compress the protocol header stack, UDP/L2TP/PPP. Profile 2
   compresses only L2TP data messages. It does not compress L2TP
   control messages.

   Profile 3: This profile is a variant of profile 2. This profile is
   defined for L2TP/PPP compression, i.e., compression of the UDP header
   is not attempted when UDP protocol is being used to traverse a NAT.
   The advantage of this profile is that it can be used to traverse
   firewalls and NATs.

   Profile 4: For GRE based tunnels. For example, for GRE over IP
   tunnel with protocol header stack IP/GRE, this profile can be used to
   compress GRE header.

5.2. TuCP Packets and Packet Types

   As mentioned in section 4, in order to compress the header, TuCP
   classifies the header fields based on how their values change during
   a flow. These fields are classified and assigned to the static and
   dynamic chain of the compressed header packets. TuCP defines two
   different header types; Initializing-Static Dynamic (IN-SD) and
   Compressing-Dynamic (COMP-D). Figures 8 and 9 show the structure of
   general header format of TuCP packets. TuCP uses these two packet
   types to establish the information in the decompressor. First, the
   static and dynamic information are sent to the decompressor, and
   after that only the dynamic information or its compressed value is
   sent to the decompressor. The static information is sent only in
   IN-SD packets. The dynamic information is sent in both packet types.

   NOTE: In COMP-D packets, dynamic fields MAY be encoded to obtain
   further improvement in terms of compression efficiency.

   For each TuCP profile, the static and dynamic fields will be composed
   of different header fields according to the stack of headers forming
   the tunneling headers. The TuCP payload (data) consists of the
   tunneled IP packet plus its payload that was initially tunneled by
   the remote host to be carried into the tunnel. TuCP does not
   compress the payload and it is transmitted as it is to the other
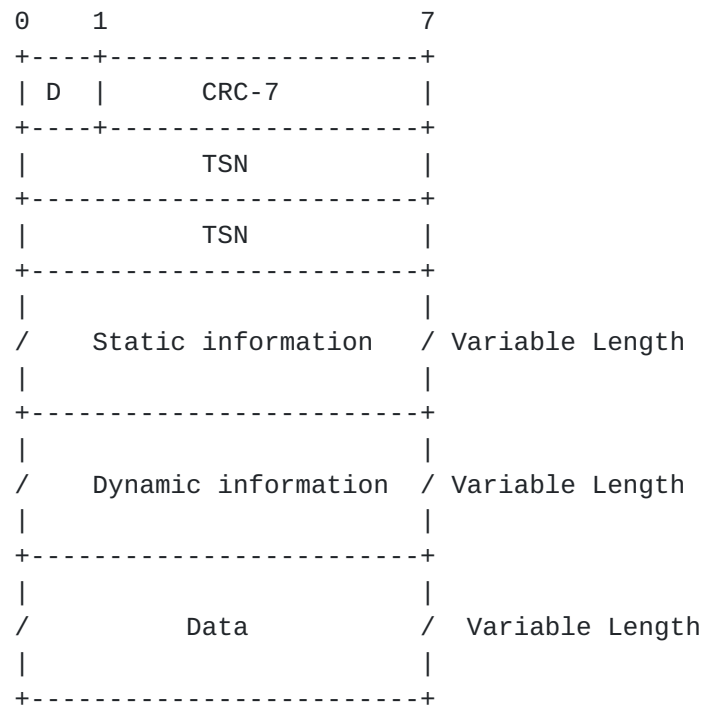   endpoint.

```
                 0    1                   7
                 +----+--------------------+
                 | D  |       CRC-7        |
                 +----+--------------------+
                 |           TSN           |
                 +-------------------------+
                 |           TSN           |
                 +-------------------------+
                 |                         |
                 /    Static information   / Variable Length
                 |                         |
                 +-------------------------+
                 |                         |
                 /    Dynamic information  / Variable Length
                 |                         |
                 +-------------------------+
                 |                         |
                 /           Data          /  Variable Length
                 |                         |
                 +-------------------------+
```

       Figure 8: TuCP General Header Format Packets: IN-SD Packet


```
                 0    1                   7
                 +----+--------------------+
                 | D  |       CRC-7        |
                 +----+--------------------+
                 |           TSN           |
                 +-------------------------+
                 |           TSN           |
                 +-------------------------+
                 |                         |
                 /   Dynamic information   /  Variable Length
                 |                         |
                 +-------------------------+
                 |                         |
                 /           Data          /  Variable Length
                 |                         |
                 +-------------------------+
```
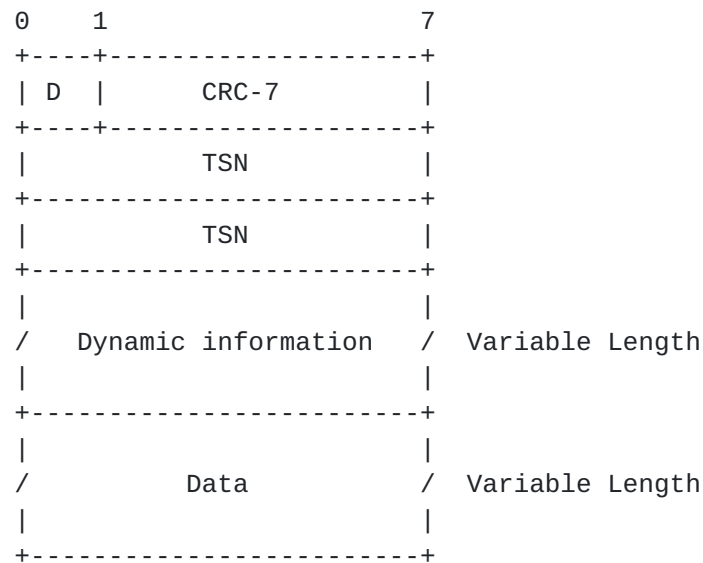
       Figure 9: TuCP General Header Format Packets: COMP-D Packet

   A description of the fields present in the TuCP header is given
   below, in the order of their appearance in the header as shown in
   figures 8 and 9.

   D (Description Type Bit): It is a 1-bit field. The value of D-bit is

interpreted as shown in Figure 10. The D bit indicates the type of
TuCP header format; IN-SD or COMP-D.

```
                +---------+-------------+
                | D-bit   | Packet Type |
                +---------+-------------+
                |   0     |    IN-SD    |
                |   1     |   COMP-SD   |
                +---------+-------------+
```

Figure 10: Description (D) type bits

CRC-7: This is a 7-bit field. The CRC (Cyclic Redundancy Check)
covers the entire original header (tunneling header). A 7-bit CRC is
computed over the TuCP header, the static fields, and the dynamic
fields, before compression. Similarly, CRC is computed at the
decompressor side, after decompression. If the CRC check is
successful, the decompressor can update its header fields'
information previously stored, using the information in the TuCP
packet received. The CRC computation in TuCP is dependent on TuCP
packet type. For IN-SD packet, CRC is computed over the entire
original header (TuCP header, static fields, and dynamic fields).
For COMP-D packet, CRC is computed over TuCP header plus original
tunneling header. The CRC coverage MUST include TuCP header because
TSN can be wrong in the TuCP header at the decompressor side. It
should be noted that since CRC bits are part of TuCP header itself,
therefore in order to compute CRC, first we MUST set all bits of CRC
field to zero, and then we should compute CRC over the header.

TSN (Transfer Sequence Number): It is a 16-bit field. This field is
introduced in the TuCP header to tackle the problem of disordering of
packets. TSN gives the decompressor the transmission order in which
packets have been sent (by the compressor) and hence allows
identifying out of order packets. The value of TSN is incremented
with every packet sent. This field is used by the decompressor to
detect the loss of packets or reordering (in a packet flow).

In addition, static and dynamic chains are added to the above fields
according to the (tunneling) protocol to form the rest of the TuCP
packet. The size of static and dynamic chains is variable. The data
field is also variable.

6. TuCP Negotiation

The first phase in the TuCP protocol is negotiation of parameters

between the tunnel endpoints. During negotiation, the TuCP
compressor and decompressor learn about the different characteristics
of the connection (tunnel or link) and the parameters that will be
used for compression. This negotiation is done during tunnel set up
between the two endpoints. Presently, TuCP protocol operates in
unidirectional mode (U mode), which implies that there is no feedback
from the decompressor to the compressor end.

The TuCP parameters are classified as static-parameters (long term-
parameters) and dynamic-parameters. Static parameters are those
which do not change for a long period of time. Therefore, these
parameters are negotiated during tunnel set up and are used during
life time of one tunnel. On the other hand, dynamic parameters are
those which change quite often, for a flow or packet. Thus, dynamic
parameters (for example, TSN) are sent in TuCP header fields.

The negotiation of some of the parameters like MRU (Maximum Receive
Unit), MTU (Maximum Transmission Unit), and MRRU (Maximum Received
Reconstructed Unit) between the tunnel endpoints SHOULD be done at
the tunnel level itself.

TuCP negotiation is profile dependent. Each TuCP profile will handle
negotiation process itself. For example, in case of profile 2,
negotiation will be done through (exchange of) L2TP control messages
between tunnel endpoints. It should be noted that TuCP does not
compress L2TP control messages. It compresses only L2TP data
messages. The following parameters MUST be configured or established
during TuCP negotiation which is one of the steps of tunnel set up
process:

TuCP-Profile: This parameter indicates a profile supported by both
the compressor and decompressor. Each profile has a different set of
static and dynamic fields. For each TuCP profile, the static and
dynamic fields will be composed of different header fields according
to the stack of headers forming the tunneling headers. The
decompressor needs to know the profile used in compression in order
to know the header format of the received (TuCP) packet. The
compressor MUST NOT compress using a profile that is not defined in
TuCP profiles. Presently, five profiles are defined in TuCP as shown
in Figure 7. The profile does not change for a tunnel between two
nodes during tunnel life time. The tunnel flow will remain the same
for a tunnel type, for example, for an IP over UDP tunnel, we will
always use TuCP profile 1 to compress the UDP headers. Thus, this
parameter SHOULD be negotiated during tunnel set up for a tunnel
type.

Inner-HC (Inner-HeaderCompression): TuCP can be used in conjunction
with an existing header compression protocol where the latter is used

(optionally) to compress the inner IP headers (inner encapsulation or
tunneled header) of IP packet carried into the tunnel. The
parameter, Inner-HC is configured during the TuCP negotiation and it
identifies the compression type (for example, ROHC, VJCOMP, IPHC,
CTCP, ECRTP) for the inner header compression. The use of inner
header compression is OPTIONAL. The use of Inner-HC and its type
should be negotiated during TuCP negotiation. When the inner header
compression is used, its compression parameters SHOULD be negotiated
during the TuCP negotiation itself. For example, if compression type
for inner headers is ROHC, then ROHC parameters are negotiated during
TuCP negotiation.

The mobile/NEMO network scenario considered in this draft considers
one tunnel flow during entire tunnel life time. However, in the core
network scenario, there can be more than one tunnel flows. In the
later scenario, TuCP establishes a context at both the endpoints of
the tunnel to achieve a successful compression and decompression of
packet headers. Each flow has its own compression context on the
compressor side and decompression context on the decompressor side.
A Context Identifier (CID) should be used to identify the context
used to compress and decompress the packet. In this case, a CID
field SHOULD be appended to the TuCP header. The size of CID field;
small or large CID SHOULD be negotiated during tunnel set up. The
CID field and its size is out of the scope of this draft as it
considers the use of TuCP in a mobile network scenario, where CID is
not used.

7. TuCP Compression and Decompression

First, there is a negotiation of static-parameters such as TuCP-
Profile and Inner-HC between the compressor and decompressor (tunnel
endpoints) during tunnel set up. Then, the compressor sends the
static and dynamic information to the decompressor. The subsequent
packets are compressed (and then decompressed) using this complete
header information stored at the tunnel endpoints.

At the compressor side, once a tunneling packet is received, the
tunneling headers are compressed using the TuCP profile negotiated
during the tunnel set up. This generates a TuCP packet which is sent
into the tunnel instead of the original (input) packet.

At the decompressor side, when the decompressor receives a TuCP
(compressed) packet, it decompresses the compressed packet and
regenerates the original packet. The decompressor MUST use the same
profile (as supported by the compressor) for decompression and to
reconstruct the original packet. The decompressor uses CRC and TSN
checking to detect errors in the packet and to identify out-of-order

packets, respectively, as discussed below in Sections 8 and 9.

8. CRC Error Detection

The wireless and radio links have high BERs (Bit Error Rates) and
PERs (Packet Error Rates) which can generate consecutive errors in
the compressed headers and can cause loss of header fields'
information synchronization between the endpoints. TuCP uses CRC
mechanism to detect such errors on the decompression side and if CRC
check fails, it discards the packets.

TuCP uses a 7-bit CRC for error detection at the decompressor side.
At the compressor side, CRC is computed over TuCP header plus the
original (tunneling) header fields before compression. Then
decompressor verifies the CRC after decompressing the header fields
and checks whether it has received the correct information or if the
information has been corrupted due to transmission errors in the
link. Erroneous packets are dropped (i.e., not decompressed) and
only error free packets are considered by the decompressor to
complete the decompression process.

The CRC check covers TuCP header because it contains TSN (sequence
numbering) which should be included in the computation of CRC to
protect it by CRC. This is because when there is an error in TSN,
the decompressor should be able to detect it. Since, the
decompressor uses TSN to detect packet loss or reordering, it SHOULD
NOT use the corrupted TSN for this purpose.

9. Managing Packet Reordering

A significant feature of TuCP is that it is able to manage packet
reordering problem. Packet reordering [Mogul1992], [Leung07] occurs
when packets arrive in wrong order, at the destination. Due to
various reasons, such as multipath routing, and retransmissions,
packets belonging to the same flow may arrive out of order at the
destination. Such packet reordering poses performance problems.

TuCP uses a TSN (Transfer Sequence Number) field in TuCP header to
check for the order of the received packets at the decompressor side.
The decompressor keeps a record of the last received TSN. On
receiving a TuCP packet, the decompressor checks if it is in order.
If the received packet is in (the correct) order, it will be
decompressed.

TSN gives the decompressor, the transmission order in which the
packets have been sent. In case of disordering in the delivery of

packets, the decompressor has to wait until the in-order packet
arrives or a timer expires, before continuing the decompression.
When the timer expires, missing packets are assumed to be lost.
Then, they are not delivered at all, even if they eventually arrive.
While waiting for the in-order packet, an early arriving packet is
stored in a buffer. The timer and buffer are implementation
parameters.

This feature of TuCP to be able to deal with packet reordering
problem is significant since TuCP can be used in conjunction with
Inner-Header-Compression, optionally. TuCP enables the use of
existing header compression mechanisms like ROHC (for Inner-HC) which
work over an ordered delivery transmission between the compressor and
decompressor (endpoints). For example, ROHC can be used to compress
the IP packets carried into the tunnel, but ROHC [RFC3095] is
designed to work over an ordered delivery transmission between the
endpoints and it does not support packet reordering. A solution for
this problem has been suggested in [RFC4224] which supports
disordered delivery of packets. However, this solution reduces
robustness of ROHC, thereby reducing the performance of ROHC over
wireless links. TuCP provides a solution to deal with packet
disordering problem, which does not reduce the performance of ROHC or
any other inner header compression and at the same time delivers
packets in order.

10. IANA Considerations

This document defines a new IP protocol for tunneling header
compression. It requires a protocol number to be attributed by IANA.

11. Security Considerations

This document by itself does not add any security risk to the use of
header compression as they have already been defined in each
mechanism.

12. References

12.1. Normative References

[BCP]   Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels, BCP 14", RFC 2119, March 1997.

[GRE]   Farinacci, D., Li, T., Hanks, S., Meyer, D., and P.
           Traina, "Generic Routing Encapsulation", RFC 2784,

March 2000.

[L2TP]   Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn,
            G., and B. Palter, "Layer Two Tunneling Protcol",
            RFC 2661, August 1999.

[PPP]   Simpson, W., "The Point-to-Point Protcol", RFC 1661,
            July 1994.

[UDP]   Postel, J., "User Datagram Protocol", RFC 768,
            August 1980.

12.2. Informative References

[CTCP]   Casner, S., Jacobson, V., and B. Thompson, "Compressing
            IP/UDP/RTP Headers for Low-Speed Serial Links", RFC 2508,
            February 1999.

[ECRTP]   Koren, T., Casner, S., Geevarghese, J., Thompson, B., and
            P. Ruddy, "Enhanced RTP (CRTP) for Links with High Delay,
            Packet Loss and Reordering", RFC 3545, July 2003.

[I-D.softwire-hs]
            Storer, B., Pignataro, C., Santos, M., Stevant, B., and J.
            Tremblay, "Softwire Hub & Spoke Deployment Framework with
            L2TPv2", draft-ietf-softwire-hs-framework-l2tpv2-12.txt
            (work in progress), March 2009.

[IPHC]   Degermark, M., Nordgren, B., and S. Pink, "IP Header
            Compression", RFC 2507, February 1999.

[Leung07]   Leung, K. and D. Yang, "An Overview of Packet Reordering
            in Transmission Control Protocol (TCP): Problems,
            Solutions, and Challenges", 2007.

[MobileIP]
            Perkins, C., "IP Mobility Support", RFC 2002,
            October 1996.

[Mogul1992]
            Mogul, J., "Observing TCP dynamics in real networks",
            1992.

[NEMO]   Devarapalli, V., Wakikawa, R., Petrescu, A., and P.
            Thubert, "Network Mobility (NEMO) Baisc Support Protocol",
            RFC 3963, January 2005.

[RFC4224]   Pelletier, G., Jonsson, L-E., and K. Sandlund, "Robust

Header Compression (ROHC): ROHC over Channels That Can Reorder Packets", RFC 4224, Jan 2006.

[ROHC]   Bromann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, L-E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., and H. Zheng, "Robust Header Compression (ROHC): Framework and four profiles: RTP,UDP,ESP, and uncompressed", RFC 3095, July 2001.

[RTP]   Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[TCP]   Postel, J., "Transmission Control Protocol", RFC 793, September 1981.

Authors' Addresses

   Priyanka Rawat
   TELECOM Bretagne
   2 rue de la Chataigneraie
   CS 17607
   35576 Cesson-Sevigne Cedex
   France

   Fax: +33 2 99 12 70 30
   Email: Priyanka.Rawat@telecom-bretagne.eu

   J-M Bonnin
   TELECOM Bretagne
   2 rue de la Chataigneraie
   CS 17607
   35576 Cesson-Sevigne Cedex
   France

   Fax: +33 2 99 12 70 30
   Email: jm.bonnin@telecom-bretagne.eu

Ana Minaburo
JCP-Consult
Cesson-Sevigne Cedex
France

Email: ana.minaburo@jcp-consult.com

Eun Kyoung Paik
KT
Central R&D Lab. KT
Korea

Email: euna@kt.co.kr